Controlling Flow Reconfigurations in SDN

Stefano Paris, Apostolos Destounis, Lorenzo Maggi, Georgios S. Paschos, and Jérémie Leguay
Mathematical and Algorithmic Sciences Lab
France Research Center - Huawei Technologies Co. Ltd.
Boulogne-Billancourt, France
Email: {name.surname}@huawei.com

***Abstract*—Software-Defined Network (SDN) controllers include mechanisms to globally reconfigure the network in order to respond to a changing environment. While iterative methods are employed to solve flow optimization problems, demands arrive or leave the system changing the optimization instance and requiring further iterations. In this paper, we focus on the general class of iterative solvers considering an exponential decrease over time in the optimality gap. Assuming dynamic arrivals and departures of demands, the computed optimality gap at each iteration $Q(t)$ is described by an auto-regressive stochastic process. At each time slot the controller may choose to apply the current iteration to the network or not. Applying the current iteration improves the optimality gap but requires flow reconfiguration which hurts QoS and system stability. To limit the reconfigurations, we propose two control policies that minimize the flow allocation cost while respecting a network reconfiguration budget. We validate our model by experimenting with a realistic network setting and using standard Linear Programming tools used in the SDN industry. We show that our policies provide a practical means of keeping the optimally gap small within a given reconfiguration constraint.**

I. Introduction

Software-Defined Network (SDN) architectures unleash the potential to compute routing at a powerful central controller and then reconfigure the network accordingly in real-time [1]. An SDN controller centrally decides traffic engineering (TE) rules to meet performance requirements such as QoS and resilience, which mirrors the past TE techniques [2] but with a new global and online twist. To maintain the best network flow configuration, the SDN controller has to solve variants of the classical Multi-Commodity Flow (MCF) problem [3] in real-time, which may involve millions of variables and constraints in large networks. As the problem instance itself evolves over time due to time-varying demands, the SDN controller solves a sequence of routing problems and needs to constantly reconsider the flow configuration. Finally, to satisfy application requirements, the controller needs to solve these problems under tight timing constraints.

To cope with the above challenges, state of the art one-shot approaches propose methods to schedule routing configurations in a predictive manner [4]. Alternatively, research ideas from the community of online algorithms [5] can be leveraged to yield a static configuration that fares well in the future when unknown demands have arrived. In this work we depart from the static approach and we propose a purely dynamic one. New demands are tentatively treated in a suboptimal way in order to meet timing requirements. Then the controller continuously re-computes global routing and reconfigures the network from time to time in order to maintain low running cost.

In particular, this paper considers a general class of iterative routing optimization solvers which yield a sharp improvement of the objective function during the very first iterations and exhibit *diminishing returns*, in the sense that the smaller the optimality gap, the longer it takes to improve it. In our dynamic setting, while the iterative solver converges to the solution, new demands arrive and old demands leave the system, changing the instance of the optimization. These considerations lead naturally to an autoregressive model for the optimality gap, whereby the gap decreases exponentially by the solver and increases whenever the demands change.

There might be a discrepancy between the computed solution at the SDN controller and the actual flow configuration in the network. At each time slot the SDN controller has the option to reconfigure the network flows as per the current computed solution. However, flow reconfigurations degrade QoS and introduce inertia into the system, hence we may avoid them and neglect to apply the most recent computed configuration. On the other hand, using the existing configuration may quickly become costly for the network since the new flows are configured with quick suboptimal decisions. In this paper we study *the fundamental problem of online SDN controllers, to decide when to perform flow reconfigurations*.

To provide an answer we formulate a stochastic optimization problem where we want to minimize the actual network cost by selecting the reconfiguration instances subject to a budget. The budget refers to a time-average constraint on the frequency of network reconfigurations, so that no more than $h_{max} < 1$ network reconfigurations occur per iteration of the solver. Thinking of the reconfiguration as sampling, the problem refers to sampling an autoregressive process with a given sampling frequency so that the extra surcharge incurred by non-sampled instances is minimized.

We first restrict ourselves to a subclass of control policies that always "sample" after the optimality gap has increased. This constraint leads to a renewal structure, which in turn permits the characterization of the best policy of the subclass. The policy works in two levels: (i) a virtual queue captures the price of sampling as it evolves over renewal frames, and (ii) a dynamic programming method is used to find the optimal "sampling" within a frame subject to the current price. Finally, we validate our system model on a realistic network scenario and compare the performance of the optimal renewal policy to a greedy non-renewal method which minimizes a drift-plus penalty function at each instance.

II. System Model

In this section, we present the routing system model that we consider.

A. System Architecture

We consider an online routing system with two main stages 1) a stage where we accept new demands and 2) a stage where we re-consider flow configuration over time. The target is to minimize additive cost which is motivated in the domain of datacenter interconnection or enterprise networks, where the goal is leasing the cheapest connections from Internet Service Providers.

Fast Connection Setup (FCS). When connection requests arrive at ingress nodes, the controller finds a feasible path satisfying multiple constraints (e.g., capacity, and QoS). For QoS purposes, the time requirements for finding a solution might be very strict. Hence at this stage, the goal is not to optimize the network, but rather to find a quick feasible solution. Example of FCS methods include (constrained) shortest path algorithms which run on residual graphs.

Garbage Collection (GC) of network resources. The sequence of sub-optimal network configurations obtained from FCS, poses significant concerns on the evolution over time of the global objective function. Therefore, periodic or event-based reconfiguration of the overall flow can help reduce the optimality gap. We call this mechanism *Garbage Collection (GC) of network resources* since it mirrors the way a Java virtual machine collects garbages and reorganizes the memory. Example of GC methods include algorithms that solve the minimum cost MCF problem.

Fig. 1 shows an example of an SDN controller which strives to minimize cost. The FCS uses a shortest path algorithm, while the GC uses a min-cost MCF solver. The numbers on links indicate the link costs, while all link capacities are 1Mb/s. In the example two demands arrive subsequently. First, the red demand arrives and requires 0.5Mb/s, while the FCS allocates it to the low cost path. Then the black demand arrives and requires 1Mb/s. Since the low cost path does not have enough capacity, the FCS allocates the black demand to the high cost path. Finally, the controller runs GC and decides to swap the two demands, saving in this way 33% of the cost.

B. Min-Cost Optimization

In the following, we turn our attention to the GC step and explain how this mechanism can be implemented.

We model the network infrastructure with an undirected graph $G = (\mathcal{N}, \mathcal{L})$, where set $\mathcal{N}$ represents the set of network nodes and set $\mathcal{L}$ models the links $e = \{i, j\}$, connecting network nodes $i, j \in \mathcal{N}$. Each link $e \in \mathcal{L}$ has a limited capacity b_e and a cost c_e, which refer to the maximum amount of flow that can be routed and the price paid per unit of routed flow, respectively.

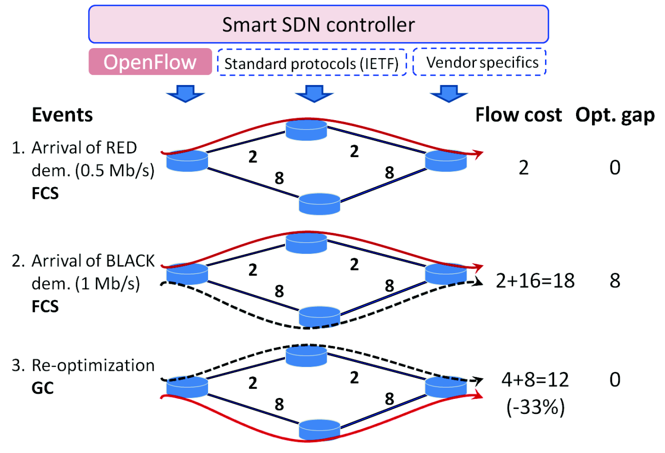


Fig. 1: Example of an online SDN routing optimization with two demands. Edge labels represent link costs. All links have capacity of 1 Mb/s. The cost evolution is computed according to the allocation performed by FCS and GC.

A unicast demand $k \in \mathcal{K}$ is identified by a source-destination pair $(s_k, d_k) \in \mathcal{N}^2$, and the amount of traffic r_k that has to be transmitted from s_k to d_k. The set $\mathcal{K}$ represents the active demands on this problem instance that need to be routed through the network. To satisfy the demands in the cheapest way, the controller needs to solve an evolving instance of the minimum cost MCF problem which can be formulated at a given time as the linear program (1)-(3), where real variables $(x_p)_{p\in\mathcal{P}}$ and $(y_e)_{e\in\mathcal{L}}$ represent the path and link utilization and take values in $[0, 1]$. In this model, P is the set of all network paths, while $P_k \subseteq P$ represents the set of all paths which connects the source s_k to the destination d_k of a demand $k \in \mathcal{K}$. In contrast, the set $P_e \subseteq P$ contains all paths that use link $e \in \mathcal{L}$.

$$C^{OPT} = \min_{(x_p),(y_e)} \sum_{e\in\mathcal{L}} y_e c_e \tag{1}$$

$$\text{s.t.} \sum_{p\in P_k} x_p = 1 \qquad \forall k \in \mathcal{K} \tag{2}$$

$$\sum_{k\in\mathcal{K}} \sum_{p\in P_e : k\in P_k} x_p r_k \leq y_e b_e \qquad \forall e \in \mathcal{L}. \tag{3}$$

The objective function (1) models the overall price paid for using the network links, the constraints (2) ensure that the entire demand r_k is routed through a set of paths with routing splits $x_p r_k$, and the constraints (3) are the link capacity constraints.

C. Iterative Solver with Diminishing Returns

Due to the immense size of the problem instance (network graph and set of demands), we resort to the *Column Generation (CG)* method coupled with the simplex algorithm, which has been proposed as a key method for solving large MCF problems [6]. Such an approach is powerful because it iteratively improves the solution by considering only a small number of variables at each step. To simplify the analysis, we assume that the optimality gap is reduced exponentially fast. Our analysis applies to other iterative techniques with exponential convergence rate, including the full gradient and the stochastic gradient methods for strongly convex objective functions [7]. We leave as future work the analysis of solution techniques with sublinear and linear convergence rates [8], [9].

At every iteration of our solver we obtain a feasible flow solution with cost $C(t) \geq C^{OPT}(t)$. Let us denote the optimality gap with $Q(t)$:

$$Q(t) \triangleq C(t) - C^{OPT}(t) \geq 0.$$

$Q(t)$ is an indication of the amount of surcharge an operator needs to pay for not having the network completely optimized at time t, hence we would like $Q(t)$ to be as small as possible. Although the iterations monotonically decrease the optimality gap $Q(t)$, we observe "diminishing returns", in the sense that the smaller the gap, the longer it takes to improve it. To model this situation we assume that the evolution of the optimality gap can be captured by the following equation:

$$Q(t+1) = (1-\rho)Q(t), \tag{4}$$

where $\rho \in (0,1)$ is a constant that relates the volume of the next improvement to current optimality gap values. This corresponds to an exponential improvement function of the type $(\frac{1}{1-\rho})^{-t}, t = 1, 2, \ldots$.

We validate our model (4) using the CG solver in different traffic conditions in the GEANT network topology [10]. The traffic matrix is generated by randomly selecting source-destination pairs and rescaling their demand according to the capacity of the bottleneck link. Fig. 2 shows the evolution of the optimality gap as a function of the number of iterations. Specifically, red and black solid lines represents 50 and 99 percentiles obtained over 500 simulations. The optimality gap $Q(t)$ obtained in numerical simulations is always lower and upper bounded by two exponential functions, namely 9^{-t} and 2^{-t}, which are depicted as green and blue dashed lines in the figure.

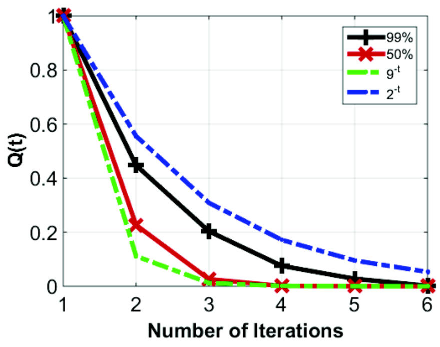


Fig. 2: Evolution of the optimality gap Q(t) in the GEANT scenario. The 50 and 99 percentiles are computed considering 500 simulations.

D. Modeling Event Arrivals

In the above section we defined the MCF instance for a given set of demands $\mathcal{K}$. Next, we consider the arrival of "new events" in the system which potentially lead to a different set of demands $\mathcal{K}(t)$. The new events correspond to arrivals of new demands, or departure of old demands and in both cases they result in a "jump" in the optimality gap $Q(t)$, due to the suboptimal network status resulting from the occurred event [1]. To simplify the considerations here, we assume that all events incur the same "jump" (i.e., the same extra cost), which is denoted by e. Our work can be easily extended to consider more elaborate models. More precisely, we assume that new events (arrivals and departures of demands) occur according to an i.i.d. stochastic process $A(t)$ with mean $E[A(t)] = \lambda$, and variance $\text{Var}[A(t)] = \sigma_A^2$, both finite.

The controller will perform the t iteration of the GC procedure by solving the $\mathcal{K}(t)$–instance of the optimization (1)-(3). The optimality gap evolution can now be rewritten to include the addition of the cost due to changing demands

$$Q(t+1) = (1-\rho)Q(t) + eA(t). \tag{5}$$

This is a first order auto-regressive stochastic process with discrete non-Gaussian disturbance. It is known that $Q(t)$ is strongly stable as long as $\rho > 0$. Conclusively, the optimality gap $Q(t)$ remains finite irrespective of how high the arrival rate of events is, or how slow the exponential slope of our solver is. We can derive the stationary mean and variance:

$$\overline{Q} = \lim_{t\to\infty} \mathbb{E}\left[Q(t)\right] = \frac{e\lambda}{\rho},$$
$$\sigma_Q^2 = \lim_{t\to\infty} \text{Var}[Q(t)] = \frac{e^2\sigma_A^2}{\rho(2-\rho)}.$$

However, to achieve this performance, the SDN controller must reconfigure the network at each iteration. The objective of the next sections is to derive control policies that yield small optimality gap without reconfiguring continuously the network.

III. Control Problem Formulation

Flow reconfigurations take time and cause small disturbances that affect the QoS, hence we would like to minimize them. However, the goal of minimizing flow reconfigurations conflicts with the goal of minimizing the average error. For example, always applying the new solver iteration yields the best possible average optimality gap $\overline{Q}$, but incurs at the same time the maximum number of reconfigurations (one every iteration). On the other hand, we may decide to periodically reconfigure once every ten iterations, which limits the reconfiguration frequency to 10% but results in operational cost higher than $\overline{Q}$, since in many iterations improved solutions are available but not applied to the network.

We consider a controller which at each slot decides whether to use the improved solution of the iterative solver or not. By selecting $u(t) = 1$ the controller decides to "spend" one

[1]A newly arriving demand is treated by the FCS procedure. We will assume that such a path can always be found. This assumption is not restrictive in practice because (i) systems are often overprovisioned, and (ii) in the case of a network overload, a congestion controller may reject some demands to make the system feasible.

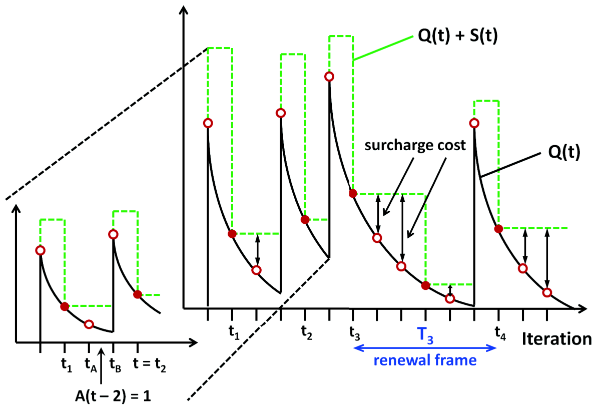


Fig. 3: Evolution of the optimality gap obtained using the MCF solver (red points) and the control policy with renewals (dashed green curve). The black curve is the interpolation of the points computed by the MCF solver at each iteration. Solid dots show where the solution computed by the solver is applied.

reconfiguration to bring the network into a form that agrees with the current output of the iterative solver. If $u(t) = 0$ is selected, then the network is left untouched.[2]

While the solver's solution has an optimality gap $Q(t)$ which evolves according to (5), the actual operational optimality gap is higher when we do not apply the best solution at each iteration, since the distance from the optimal point increases as illustrated in Fig. 3, where the *"surcharge cost"* between the solid black line (the output from the solver) and dashed green line (the cost of the current network configuration) keeps increasing as long as the $u(t) = 0$. We denote by $S(t)$ the surcharge cost at slot t caused by not using the most recent improved solution. The evolution is given by

$$S(t) = (S(t-1) + \rho Q(t-1))\,(1 - u(t)), \tag{6}$$

where we note that (i) if $u(t) = 1$ then $S(t)$ becomes zero, while (ii) if $u(t) = 0$ then $S(t)$ increases by a term $\rho Q(t-1)$ which is the new cost improvement computed by the solver but not applied to the network.

The objective is to find a control policy that selects $u(t)$ at each slot in order to minimize the average surcharge cost subject to a constraint on the average number of flow reconfigurations (i.e., the number of times we select $u(t) = 1$). We may formalize a stochastic optimization problem as follows.
Minimum Surcharge subject to Reconfiguration Frequency:

$$\text{Minimize } \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{S(t)\} \tag{7}$$

$$\text{s.t. } \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{u(t)\} \leq h_{max}. \tag{8}$$

where $0 < h_{max} \leq 1$ is the constraint on the frequency of slots where we reconfigure. There exist potentially other objectives or constraints with which we would like to generalize problem (7), such as for example to minimize the number of routers that become reconfigured. In this paper we focus on the solution of (7) which is fundamental for SDN controllers.

[2]Without loss of generality, we leave for future work other granularities of reconfiguration like per-device and per-flow.

Problem (7) is a stochastic online optimization that admits a class of policies Π. Here, we are interested in causal policies, that is policies with no knowledge/prediction about the time instances of future arrivals. In more detail, at the beginning of slot t, the controller is given the optimality gap $Q(t)$ as computed by the solver. Additionally, the controller knows the past evolution of the system, that is the values $(A(\tau), Q(\tau)), \forall \tau < t$. Any new flow arrivals/departures that occur at slot t are taken into account for computing the (new) optimal flow allocation from slot $t+1$ and onwards. The system is then Markovian with state vector

$$\mathcal{S}(t) = [Q(t), S(t-1), Q(t-1)]$$

(the last two state variables are needed to find the cost (6) at each slot) and the problem is a constrained Markov Decision Process (MDP) with infinite horizon and time-average cost and constraint, where the system dynamics and the cost at every slot are given by (5) and (6). It is known [11, Th. 6.2] that we can restrict to the set of Markov controls (i.e. one that makes the control decision at slot t based only on $\mathcal{S}(t)$ and knowledge of statistics of the arrivals) without any loss of optimality; this means that policies that use older information or are allowed to depend on the time slot index do not lead to a better performance than those that make decisions based only on the current state of the system and its statistics. Nevertheless, since (7) includes the solution of a constrained MDP with infinite state space, it is impossible to solve efficiently using standard methods. In the next section we restrict our attention to a subclass of policies that invoke a renewal structure and include an efficient solution.

IV. Control Policy with Renewals

We say that a Markov process has a renewal structure if the system visits states where it "*statistically restarts*", cf. [12, Ch. 7]. Such systems are much easier to analyze since the online problem can be broken down to smaller control problems within each renewal frame. Unfortunately, the problem (7) we are interested in does not have such a renewal structure. Our approach in this section is to use an artificial constraint to invoke a renewal structure into our system. We will later show that subject to this constraint, there is an efficient policy that optimizes (7), thus providing a complete performance characterization of the system under this constraint.

Policy Constraint 1 (Reconfigure after Demands Change): Consider the constrained set of policies $\Pi_c \subset \Pi$. For any policy $\pi \in \Pi_c$ the reconfiguration is applied whenever there was a change in the demands, i.e., at any t

$$u(t) = 1 \quad \text{if} \quad A(t-2) > 0, \quad \forall t, \quad \forall \pi \in \Pi_c.$$

The delay of 2 slots in $A(t-2)$ relates to our notation and ensures that the controller is forced to reconfigure at the first iteration after the arrivals have been considered, see the blow-up box of Figure 3. In the figure, $A(t_A)$ represents arrivals in $[t_A; t_B)$ that are soon handled by FCS using the network configuration in state $\mathcal{S}(t_A)$. These arrivals are handled by GC at time $t_{A+1} = t_B$ (i.e., their paths are integrated in the problem (1)-(3) at t_{A+1}) and only at $t_A + 2$ the benefit of GC is available. Intuitively the constraint is justified since the first step of the solver has the steepest decrease in the optimality gap and hence by choosing $u(t) = 1$ in such slots a great extra cost is avoided.

Note that for Π_c to be feasible, it must be $\mathbb{P}\{A(t) > 0\} < h_{max}$. If otherwise, the reconfiguration constraint (8) will be violated by any policy in Π_c.

Under a policy $\pi \in \Pi_c$ the time interval between two consecutive slots with the property "$A(t-2) > 0$" constitutes a renewal frame. Let $t_0 = 0$, denote t_n the $n-$th slot satisfying $A(t-2) > 0$, and denote $T_n = t_{n+1} - t_n$ the number of time slots in the $n-$th frame (note that T_n is a random variable).

In the remaining of this section we propose RP (Renewal Policy), a policy that satisfies the Constraint 1 and hence RP $\in \Pi_c$. By exploiting the renewal structure that policies in Π_c induce, we will prove that RP is a feasible policy that achieves near-optimal value of (7) over all policies in class Π_c.

A. *Renewal Policy (*RP*)*

A standard way to solve stochastic optimization problems involves the use of policies that greedily balance the penalty in a time slot with the instability of virtual queues; see the *drift-plus-penalty algorithm* [12]. In fact, this dynamic algorithm can also be applied to systems with renewals. In this case, the policy keeps track of the budget spent thus far and puts a price per unit of budget to be spent in the next renewal frame; in our system we have a price per reconfiguration. This budgeting method guarantees long-term feasibility. Within the renewal frame, an oracle policy is used to decide the control taking into account the price per reconfiguration. When the uncontrolled states of the system are i.i.d. renewals, the problem is tackled in [12], [13]. Here we extend it to our non-i.i.d. framework.

To introduce a "price" into our system we define a virtual queue $U(t_n)$ whose value we track only at the end of each renewal frame:

$$U(t_{n+1}) = \left[U(t_n) - T_n h_{max} + \sum_{t=t_n+1}^{t_n+T_n} u(t) \right]^+, \qquad (9)$$

where the term $\sum_{t=t_n+1}^{t_n+T_n} u(t)$ equals the number of reconfigurations used in the last renewal frame, while $T_n h_{max}$ is the product of the last renewal frame length T_n times the average constraint on reconfigurations from (8). From Queueing Theory point of view, the implications here are clear. Mean rate stability [12, p. 17] of $U(t_n)$ guarantees the property "arrivals $\leq$ departures", hence

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{u(t)\} \leq h_{max}$$

which satisfies (8). Hence, any policy that stabilizes $U(t_n)$ is a feasible policy. To track in real-time the stability of $U(t_n)$, we define the quadratic Lyapunov drift as

$$\Delta(\mathbf{s}, U(t_n)) = \mathbb{E}\left[U^2(t_n + T_n) - U^2(t_n) \; |\mathbf{S}(t_n) = \mathbf{s}, U(t_n) \right]$$

For a given policy, $\Delta(\mathbf{s}, U(t_n))$ measures how $U(t_n)$ drifts over a renewal frame. Any policy that yields bounded $\Delta(\mathbf{s}, U(t_n))$ for any $(\mathbf{s}, U(t_n))$ can be shown to stabilize $U(t_n)$ in the mean rate sense, hence such a policy is feasible.

Apart from feasibility, we are also interested in minimizing the cost (7), thus we combine the drift $\Delta((\mathbf{s}, U(t_n))$ with the cumulative penalty (7) within a renewal frame:

$$DPP(\mathbf{s}, U(t_n)) = \Delta(\mathbf{s}, U(t_n)) + V\mathbb{E}\left\{ \sum_{t=t_n+1}^{t_n+T_n} S(t) \right\}, \qquad (10)$$

where V is a constant that weighs the two goals. The metric (10) is often called *Drift-Plus-Penalty* (DPP). Minimizing drift-plus-penalty includes two conflicting goals, (i) to minimize the drift $\Delta(\mathbf{s}, U(t_n))$ thereby satisfying in the long term the constraint (8), or (ii) to greedily minimize the penalty in the next renewal frame. In fact, $u(t) = 0$ favors (i) and $u(t) = 1$ favors (ii). A policy that minimizes drift-plus-penalty at each renewal frame is essentially striking a good balance between the two in a greedy fashion.
If we perform standard calculations and expand (10) we obtain an upper bound expression on $DPP(\mathbf{s}, U(t_n))$ which is optimized at every renewal by a policy that solves the following optimization problem:

$$J^*(t_n; V) = \min \mathbb{E}\left\{ \sum_{t=t_n+1}^{t_n+T_n} VS(t) + U(t_n)u(t) \right\} \qquad (11)$$

This optimization seeks to find an appropriate sequence of controls $u(t_n + 1), \ldots, u(t_n + T_n)$ within the n–th renewal frame to balance the expected price of the extra cost $VS(t)$ with the price of reconfigurations $U(t_n)u(t)$.

Next, we propose RP which is designed to minimize (11) at every renewal frame, subject to Constraint 1. The RP policy works as follows. Following a slot with an arrival, it always selects $u(t) = 1$ and notes the beginning of a renewal frame. The virtual queue (9) is used to track the evolution of the price across renewal frames. At the beginning of the n–th renewal RP observes the value $U(t_n)$ and calls a routine ϵ–OPT$(U(t_n))$ which approximately solves (11). The routine returns an infinite sequence of controls $u(t_n{+}1), u(t_n{+}2), \ldots$. RP uses this sequence until the n–th renewal is over at which point it discards the remain subsequence and starts over. The policy is described in the next page.

B. *Performance Analysis of* RP

In this subsection we build intuition about why RP is optimal in class Π_c. The technical proof of the result will be presented in our followup work.

1) $Q(t)$ is an ergodic Markov chain: For the analysis we will make the following assumption

Renewal Policy (RP)

Parameter Selection. $t_0 = 0, V = 100, U(t_0) = V$.

Renewal frame. Check every slot. If at slot $t-2$ we had $A(t-2) > 0$, then slot t is the beginning of a renewal frame. This recursively defines a sequence of time instances $t_0, t_1, \dots, t_n, \dots$ that indicate the beginning of renewals, where the slots $\{t_n, \dots, t_{n+1}-1\}$ define the n–th renewal frame. Within n–th renewal the policy performs the following steps.

Step 1: Constraint 1. Choose $u(t_n) = 1$.

Step 2: Price Update. Update the price

$$U(t_n) = \left[U(t_{n-1}) - T_{n-1}h_{max} + \sum_{\tau=t_{n-1}+1}^{t_{n-1}+T_{n-1}} u(\tau) \right]^+ .$$

Step 3: Solution of (11)**.** Call routine ϵ–$\mathsf{OPT}(U(t_n))$ to obtain an infinite sequence of controls $\mathcal{U} = \{u(t_n+1), u(t_n+2), \dots\}$ that approximately solves

$$\text{Minimize } \mathbb{E}\left\{ \sum_{t=t_n+1}^{t_n+T_n} VS(t) + U(t_n)u(t). \right\}$$

We describe routine ϵ–$\mathsf{OPT}(U(t_n))$ below.

Step 4: Actions within the renewal. For time slots $\{t_n+1, \dots, t_{n+1}-1\}$ choose the controls according to $\mathcal{U}$.

Assumption 2: If $Q(t) = q_\epsilon$ for some small constant $0 < q_\epsilon << 1$, then we can approximate $Q(t) = 0$.

In practice the assumption implies that tiny optimality gaps may be disregarded, hence it is mild. Technically, it is used to prove the following intermediate result:

Lemma 3: The process $Q(t)$ evolves in a countable state space. In addition, under Assumption 2, $Q(t)$ is irreducible and aperiodic.

Proof: All proofs of claims are in the technical report. ∎

2) Optimal cost of Π_c*:* Consider the optimization (7) over all policies belonging to Π_c. The problem can still be seen as a constrained MDP, with state space $\tilde{\mathcal{S}}(t) = [Q(t), S(t-1), Q(t-1), A(t-2)]$. From the general theory of constrained Markov Decision Processes (see e.g. [11]) we can show the following:

Lemma 4: The optimal policy $\pi^* \in \Pi_c$ is one where the controller is a (possibly randomized) function of only $[Q(t), S(t-1), Q(t-1)]$ if $A(t-2) = 0$ and $u(t) = 1$ if $A(t-2) > 0$ (the latter is due to the constraint of class Π_c).

Denote the incurred cost of π^* with $\overline{S}^*$, where

$$\overline{S}^* = \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{S^{\pi^*}(t)\}.$$

Although we cannot directly characterize π^*, we will use its existence to prove the optimality of RP.

3) Analysis of RP*:* The setting we have here is slightly different with respect to optimization over renewal processes as presented in [12], [13], since $Q(t_{n+1})$ depends on $Q(t_n)$ (in the references the uncontrolled process is i.i.d. at renewal periods). In order to resolve this, we use the idea of $T-$slot drifts as employed in [12], [14] for Markovian dynamics of the uncontrolled state processes, extending it over renewal frames this time.

Specifically, define $I(t) = \mathbb{I}_{\{A(t-2)>0\}}$ and the Markov chain $\mathbf{Z}(n) = [Q(t_n), I(t_n)]$ (note that $S(t_n) = 0$ since $u(t_n) = 1 \ \forall n$, therefore $S(t_n - 1), Q(t_n - 1)$ do not matter). Then choose any of the states $\mathbf{z} = [q, 1]$. Since we also need an arrival to happen at states $\mathbf{z}$, we are essentially looking at the beginning of renewal frames of the initial problem. Starting from a point in time where $\mathbf{Z}(n) = [q, 1]$ and denoting N_q the return time to that state, we consider a variable drift for these N_q renewal frames. Assuming a routine that solves (11) with an ϵ error and standard analysis from Lyapunov optimization theory [12], we compare the drift-plus-penalty of RP to π^* to get

$$\overline{S}^{\mathsf{RP}} \leq \frac{\epsilon + B_q}{V} + \overline{S}^*. \tag{12}$$

where

$$B_q = \frac{\mathbb{E}\left\{T_n^2\right\}(1-h_{max})}{2}\mathbb{E}\left\{N_q\right\} + \frac{\mathbb{E}\left\{T_n\right\}(1+h_{max})}{2}\mathbb{E}\left\{N_q^2 - N_q\right\}.$$

For every $q \in \mathcal{Q}$ we prove that the return time has bounded second moment:

Lemma 5: $\mathbb{E}\left\{N_q^2\right\} < \infty, \forall q \in \mathcal{Q}$.

which asserts that $B_q < \infty$. Hence (12) shows that the cost achieved by RP is near-optimal (consider for example large values of V). Feasibility is shown by using the same drift analysis to prove that $U(t_n)$ is mean rate stable. Formally:

*Theorem 6 (*RP *is near-optimal in* Π_c*):* Let $J^*(t_n; V)$ be the optimal value of (11) and $\mathcal{U}$ be an $\epsilon-$optimal control policy for every renewal frame, i.e. a control policy that for every renewal frame n achieves a cost $J(t_n; V) \leq J^*(t_n; V) + \epsilon$, for a constant $\epsilon > 0$. Then, for RP the following hold:

1) The constraint of eq. (8) is satisfied.
2) $\overline{S}^{\mathsf{RP}} \leq \overline{S}^{\pi^*} + \frac{B+\epsilon}{V}$ (i.e. the average cost (7) is $(O(1/V)$ close to the optimal one, achieved by π^*), where $B = \min_{q\in\mathcal{Q}} [B_q]$ is a finite constant.

C. ϵ-optimality via Dynamic Programming

Above we required a routine that approximately solves (11). Below we describe this routine. Denote

$$\beta = 1 - \mathbb{P}\{A(t) > 0\}, \tag{13}$$

the stochastic control problem (11) with state evolution defined by (4) and (6) is equivalent to the deterministic control problem of finding a control sequence $u(t) \in \{0, 1\}, t = t_n+1, t_n+2, t_n+3, ...$ to solve

$$\text{Minimize} \sum_{t=t_n+1}^{\infty} \beta^{t-t_n} \left(VS(t) + U(t_n)u(t)\right), \tag{14}$$

where the dynamics of the system are given by (4), (6) (for $t \in \{t_n, ..., \infty\}$). To efficiently solve (14) we first note that the cost function is bounded. By exploiting this fact, we can truncate the infinite time window over which the optimal control is evaluated at time step T, and derive a truncated sequence which produces a finite discrepancy ϵ (monotonically decreasing in T) on the optimal cost. In Theorem 7 we state this formally and provide an expression of T as a function of a tolerated optimality error ϵ.

Theorem 7: Let $\epsilon > 0$. Let $\tilde{u}'$ be the optimal solution of the following truncated version of (14):

$$\tilde{u}' = \underset{u'_k}{\operatorname{argmin}} \sum_{k=1}^{T} \beta^k \left(V S_k + U(t_n) u'_k \right) \tag{15}$$
$$\text{s.t.} Q_{k+1} = (1-\rho) Q_k, \quad 1 \le k \le T-1$$
$$S_k = (1-u'_k)\left(S_{k-1} + \rho Q_{k-1}\right), \quad 1 \le k \le T-1$$
$$S_0 = 0, Q_0 = Q(t_n)$$

where the truncated time horizon T is computed as

$$T = \left\lceil \log\left(\frac{\epsilon}{(1-\beta)\left(V Q_0 + U(t_n)\right)} \right) / \log(\beta) - 1 \right\rceil .$$

The, the policy $\overline{u}' = [\tilde{u}', l_{T+1}, l_{T+2}, \dots]$, where l is any infinite binary sequence is ϵ-optimal w.r.t. the original problem in (14).

The truncated problem is a deterministic control in finite time horizon and finite state space. Below we propose a routine that uses standard Dynamic Programming methods to solve the truncated problem.

ϵ-optimal routine ϵ–$\mathsf{OPT}(U(t_n))$

Parameters. Choose discrepancy ϵ, probability of arrivals β, constant V.
Input. Price $U(t_n)$, optimality gap $Q(t_n)$.
Output. Use standard Dynamic Programming techniques (e.g. forward induction [15]) to solve (15), with T selected as given in Proposition 7. Then, the following control sequence is obtained for time slots $\mathcal{U} = \{t_n + 1, t_n + 2, \dots\}$: $u(t) = \tilde{u}'_{t-t_n}, t_n < t \le t_n + T$ and $u(t) = 0, \forall t > t_n + T$.

1) ϵ–OPT has linear complexity: We now turn to the actual computation of the ϵ-optimal strategy $\tilde{u}'$. In general, any optimal control problem with finite state space can be formulated as a Dynamic Programming (DP) technique; nevertheless, the amount of memory required by the solution of the DP problem may generally grow exponentially along the iteration step axis, which then makes the DP implementation a daunting task [15]. Fortunately, our case is tractable, since the state variables Q_k, Q_{k-1} are completely determined by the time index and initial condition and each time control $u'_k = 1$ is applied, $S_k = 0$. Therefore, looking at the corresponding Trellis tree in Fig. 4, at each time step $k+1$ there are k different paths that converge into the state $S_{k+1} = 0$ (the paths for which $u'_k = 1$). Thanks to Bellman optimality principle, we can store only the path with minimum cost and discard all the remaining $k-1$ suboptimal paths. In other words, $2k$ different paths are generated at step k, but only $k+1$ are then left for the recursion step at time $k+1$.

The dynamic programming algorithm terminates at step $k = T$, when the ϵ-optimal control always prescribes $\tilde{u}'_T = 0$ (in fact, it is easy to see that setting $\tilde{u}'_T = 0$ would incur a higher cost and does not alter the future state evolution). The path with minimum cost among the T surviving ones is selected, and the ϵ-optimal control $\tilde{u}'$ can be finally read on the labels associated to each link of the best path.

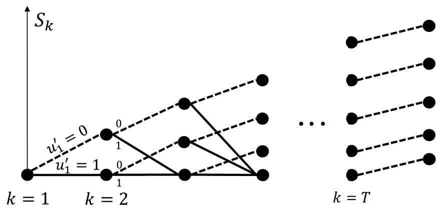


Fig. 4: Dynamic programming trellis

D. *Properties of* RP

Using Constraint 1, we converted the original problem of a constrained MDP in a very large state space to a stochastic control problem which is amenable to Lyapunov optimization solutions. In particular, it is possible to solve the constrained problem optimally by a two level approach: (i) A virtual queue is used to capture the evolution of price of reconfiguration (and monitor the long-term feasibility of the frequency constraint) and (ii) inside the renewal frame we use deterministic optimal control taking into account the price. We have shown that the latter is a feasible methodology since it can be solved in linear time to T, which is lower than the complexity of one solver iteration. Another advantage of the proposed methodology is that only the probability that a flow arrives/departs is needed and not the whole distribution of $A(t)$. Apart from $\mathbb{P}\left(A(t) > 0\right)$ which needs to be estimated, the RP policy is purely adaptive and oblivious to past events or other system statistics. Finally, we observe that the constraint is not very harmful since surcharge cost is particularly high when demands change, and hence it is desirable to reconfigure at these time instances.

V. Greedy Policy (GP)

It is tempting to propose a heuristic approach which uses the drift-plus-penalty method at every time slot instead of every renewal frame. The virtual queue that corresponds to the constraint is then updated at every slot as

$$U(t+1) = [U(t) - h_{max}]^{+} + u(t) \tag{16}$$

and we seek a policy that observes $U(t), \mathcal{S}(t)$ and tries to minimize the right-hand-side of the drift-plus-penalty expression (expectations are conditional on $U(t), \mathcal{S}(t)$ and over possible randomizations of the policy)

$$\mathbb{E}\left\{ \frac{U^2(t+1) - U^2(t)}{2} \right\} + V\mathbb{E}\left\{S(t)\right\} \le \frac{h^2_{max}}{2} + \tag{17}$$
$$U(t)(\mathbb{E}\{u(t)\} - h_{max}) + V(S(t-1) + \rho Q(t-1))\mathbb{E}\left\{(1-u(t))\right\}.$$

This can be achieved by a threshold policy of the form

Greedy Policy (GP)

$$u(t) = \begin{cases} 1 & \text{if } U(t) < V\frac{S(t-1)+\rho Q(t-1)}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Although GP minimizes the right hand side of (17), it is not a provably near-optimal policy for (7) within Π. The reason is the following: drift-plus-penalty algorithms can be proven to be near-optimal for cases where *the cost that is added at every time slot depends only on the control taken at that time slot and state variables that evolve independently of the control actions* [12]. This is not the case in our problem, since the evolution of the cost to be added at time slot t depends on the control policy taken in the previous slot, as seen by (6).

On the other hand, GP has numerous advantages. (i) Using the drift-plus-penalty we may show that it stabilizes $U(t)$ and hence it is a feasible policy. (ii) Contrary to RP, GP can be applied for constraints smaller than the probability of arrival of a flow. (iii) It is oblivious to $Q(t)$ and can be applied without knowledge of the optimal value of the optimization. (iv) It does not require any information on the statistics of how the demands change-e.g. it does not need $\mathbb{P}(A(t) > 0)$. To implement GP we only need to keep track of the virtual queue $U(t)$, and the costs $Q(t), S(t)$ from the previous slot. The constant V is chosen high enough to approximate well the optimal solution; a typical value is $V = 1000$.

VI. Numerical Results

To evaluate the performance of our control policies on a realistic online SDN routing optimization system, we have implemented a scalable algorithm based on column generation to solve iteratively the linear program (Eq. (1)-(3)). In what follows, we first describe the experimental methodology and then illustrate the results that confirm the validity of our dynamic control approach.

A. Experimental Methodology

Since we are interested in solving large MCF problems, the SDN solver we implemented works as follows. It first computes a feasible solution using FCS, allocating the demands that arrive to the system on cheapest paths over the residual graph. After this initialization phase, the solver proceeds by considering only a subset of paths and iterates by adding and removing paths (i.e., x_p variables) to a restricted version of the problem until it converges to the optimal point. At each iteration, the solver typically uses the dual formulation of Eq. (1)-(3) to add only those paths that can improve the objective function.

In order to perform the evaluation under realistic conditions, we used a real-life dataset captured in 2006 by Uhlig et al. [10] on GEANT, the high bandwidth pan-European research and education backbone. The dataset contains a topology of 22 nodes and 36 high capacity 40G links. The link cost has been rescaled in the range $[0; 100]$. We generated random traffic demands according to a Poisson process with inter-arrival time of 2 s and fixed duration of 20 s. The total simulation time is 10 min. We perform 50 independent measurements by generating as many traffic patterns, which are enough to yield very narrow confidence intervals.

B. Performance Evaluation

Fig. 5(a) illustrates the total surcharge (i.e., $\sum_{t=0}^{T} S(t)$) while 5(b) shows the reconfiguration rate $h = \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{u(t)\}$ used by the control policies as a function of the bound on the reconfiguration rate h_{max}. We compare our control schemes RP and GP against a *Periodic Policy* (PP) that consists in reconfiguring the network periodically with a period equal to $\frac{1}{h_{max}}$. Note that for RP we can only show the results for $h_{max} \geq 0.7$, which is slightly larger that the arrival rate λ in our scenario. Indeed, by definition RP needs to reconfigure at least as many times as the number of arrivals/departures.

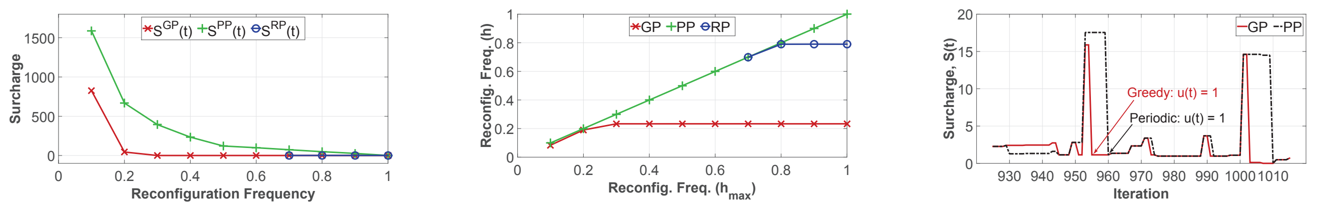
Fig. 5(a) shows that RP and GP achieve performance almost identical to continuously reconfiguring the network ($S(t) = 0$) even with a reconfiguration rate lower than 100%. Furthermore, as illustrated in in Fig. 5(b), we can observe that both RP and GP satisfy the constraint on the reconfiguration rate, namely $h < h_{max}$. However, while RP requires a reconfiguration rate slightly larger than the arrival rate to obtain such a result, the GP scheme requires a lower reconfiguration rate, thus resulting in a more economical control policy. Differently from RP, GP can select only the best iterations, namely the iterations with the largest improvement in terms of cost reduction. In our simulations, we observe that selecting the very first iterations of the column generation algorithm after an arrival/departure provides the best performance.

From the figures, it can be further observed that the PP policy performs poorly when the SDN operator has a low budget on the reconfiguration rate. Specifically, when the controller operates with a limit on the reconfiguration rate in the $[0.1; 0.4]$ interval, using a periodic policy incurs in a surcharge 3 times larger than using the GP scheme. Nonetheless, as the network and the data traffic can sustain a larger number of reconfigurations, PP provides good performance, since the time that lasts after selecting a bad iteration and the next reconfiguration decreases. In other words, even if PP selects bad iterations, it can quickly recover when the reconfiguration rate increases.

Fig. 5(c) shows a snapshot of an evolution over time of the surcharge $S(t)$ for GP (solid line) and PP (dashed line) among the 50 trials with $h_{max} = 0.1$. While PP blindly selects which iterations to use, leading to bad performance, GP "waits" until a big increase in the surcharge before applying the configuration of the solver, thus reducing the surcharge cost paid by the network operator.

VII. Related Work

SDN enables a global and online routing optimization to improve link utilization and increase reactivity to failures.



(a) Total Surcharge (b) Reconfiguration Rate (c) $S^{PP}(t)$ vs $S^{GP}(t)$ with $h_{max} = 0.1$

Fig. 5: Performance evaluation of the proposed control policies over GEANT as a function of the bound h_{max} on the reconfiguration rate.

Google has showed in 2013 that they could achieve nearly 100% of link utilization [16] with their OpenFlow WAN controller. Two main factors are behind this architectural evolution: programmable data planes and logically centralized controller platforms. Several propositions have emerged in recent years to make data plane elements programmable and offload the control logic to external units. To name a few: Forces [17], PCEP [18] and OpenFlow [19].

Recent research on routing problems has been pursued on two main lines: large problem instances and online versions of the problem. Solving large routing problems has received a lot of interest from the traffic engineering community [20]. Several approaches have been proposed using column generation to solve large problem instances [3]. However, they have been mostly developed for offline network planning tools considering the worst case scenario and do not consider their use in a dynamic routing system.

The oline version of the MCF problem, where the parameters are revealed over time, has been studied for throughput maximization or load minimization, as detailed by Even et al. [5]. In more general settings, the problem has been formulated as an online packing and covering problem [21], where the objective function as well as the packing constraints are not known in advance. While these works show sublinear competitiveness ratios, they have been mainly designed for admission control and do not propose any reoptimization of the flow allocation.

VIII. Conclusion and perspectives

Software-Defined Networking is foreseen as a means of using more efficiently network resources and dynamically adapting the routing configuration over time. In this context, this paper addresses an important question about the interplay between the high degree of configuration flexibility and the computational limits of the SDN controller logic.

Specifically, we study the evolution over time of the optimality gap of a general class of iterative routing optimization algorithms. Furthermore, we present two control policies working on top of the online routing optimization engine to decide whether to apply or not the current yet not optimal global network configuration. Numerical results show that our control schemes can easily track the evolution of the system using a bounded number of reconfigurations, thus pursuing the double objective of optimizing the performance and the system stability.

References

[1] Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617 – 1634, 2014.

[2] Ning Wang, Kin Ho, George Pavlou, and Michael Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.

[3] Kazutaka Murakami and Hyong S Kim. Optimal capacity and flow assignment for self-healing atm networks based on line and end-to-end restoration. *IEEE/ACM Trans. on Networking*, 6(2):207–221, 1998.

[4] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. *ACM SIGCOMM*, 2014.

[5] Guy Even and Moti Medina. Online multi-commodity flow with high demands. *Approximation and Online Algorithms*, 2013.

[6] Jacques Desrosiers, François Soumis, and Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984.

[7] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. *Advances in Neural Information Processing Systems*, 2012.

[8] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[9] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[10] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Review*, 36(1):83–86, 2006.

[11] Eitan Altman. *Constrained Markov Decision Processes*, volume 7. CRC Press, 1999.

[12] Michael J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

[13] Michael J. Neely. Dynamic optimization and learning for renewal systems. *IEEE Trans. on Automatic Control*, 58(1):32–46, Jan. 2013.

[14] Longbo Huang and Michael J. Neely. Max-weight achieves the exact $[o(1/v), o(v)]$ utility-delay tradeoff under markov dynamics. *arXiv preprint arXiv:1008.0200*, 2010.

[15] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific Belmont, MA, 1995.

[16] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM*, 2013.

[17] Avri Doria et al. Forwarding and Control Element Separation (ForCES) Protocol Specification, RFC 5810, IETF (March 2010).

[18] Avri Doria et al. Path Computation Element (PCE) Communication Protocol (PCEP), RFC 5440, IETF (March 2009).

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, March 2008.

[20] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic engineering in software defined networks. *IEEE INFOCOM*, 2013.

[21] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal: dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.